by
Jeff Prosise

Tutor

MORE ON SHARE.EXE

In the April 24, 1990, Tutor column, you mentioned that DOS 4.0's SHARE.EXE permits you to access drive partitions greater than 32MB. I use MS-DOS 4.01 with a 67MB hard disk set up with the entire disk serving as one DOS partition. I'm not attached to a network, and I don't use SHARE.EXE. I've had no trouble at all accessing files on the hard disk—except for the incessant message DOS displays at start-up warning me it can't find SHARE.

Can you enlighten us a little more on the subject of SHARE.EXE and large partitions? Why does Microsoft say SHARE.EXE must be loaded if you use large partitions under DOS 4.0 when, obviously, it doesn't have to be loaded?

Arthur Pfeffer
New York, New York

It's not strictly true that SHARE.EXE *must* be loaded to access large partitions under DOS 4.0. A better way to say it is that SHARE.EXE *should* be loaded—which explains why DOS 4.0 goes to great lengths to find and load SHARE.EXE if you don't load it yourself.

SHARE's original raison d'être was to arbitrate file accesses in networked environments or in any environment where two programs are likely to grab for the same file at once (as might be the case when you run TSRs that perform file I/O, for example). Used in this manner, SHARE.EXE provides file sharing and locking services and monitors every read and write access to ensure that two processes don't collide.

- **MORE ON SHARE.EXE: To be sure your files are safe, it's a good idea to load SHARE, if you use large partitions.**
- **FORMATTING YOUR DISK: Why your hard disk needs both a low-level and a high-level format.**

SHARE took on an additional function in DOS 4.0, however: ensuring that accesses to large partitions by programs using File Control Blocks (FCBs) don't inadvertently destroy data. FCBs are fixed-length data structures that were once used (before the days of file handles) to access disk files. The designers of DOS 4.0 were faced with a dilemma when they added support for partitions larger than 32MB: either alter the format of the FCB to accommodate large partitions (and render existing applications that used FCBs incompatible with the new operating system) or retain the old FCB format and do something else to keep FCB file accesses from destroying data or randomly writing data to a different file. They chose the latter, adding a module to SHARE.EXE that appends an internal file system table to an application's FCBs.

With SHARE.EXE installed, programs that use FCBs to read and write files are as compatible with partitions larger than 32MB as they are with partitions smaller than 32MB in length.

DETERMINING WHETHER SHARE.EXE IS LOADED

```
begin_test:             mov   ax,1000h
                        int   2Fh
                        cmp   al,0FFh
                        je    share_installed

share_not_installed:      .
                          .
                          .
```

Figure 1: You can use this code fragment to determine whether or not SHARE is installed. If AL is set to FFh on return from interrupt, then SHARE is installed; otherwise, it's not.

The higher-level file functions introduced in DOS 2.0 use file handles rather than FCBs to reference files. This is the preferred method for performing file I/O, but there are some popular DOS applications that still use FCBs, including *SideKick* and *dBASE III Plus*. Even COMMAND.COM gets in on the act: It uses the FCB-based interrupt 21h, function 13h when deleting files.

Unfortunately, it's difficult to tell which programs use FCBs and which do not. Thus the reasoning behind DOS 4.0's conservative approach to loading SHARE any time large partitions are used: When in doubt, assume the worst. Where the integrity of crucial files is concerned, it's better to be safe than sorry.

SHARE may be installed as a TSR either from the command line or with an INSTALL directive in CONFIG.SYS. Loading it from CONFIG.SYS ensures that it's installed early, before any other programs (or TSRs loaded from AUTOEXEC.BAT) get a chance to make FCB calls.

If your hard disk contains large partitions, DOS 4.01 automatically looks for SHARE.EXE in the directory spelled out in the SHELL= statement in CONFIG.SYS or, failing that, in the root directory of the boot drive. If it can't find it, DOS displays the message: "WARNING! SHARE should be loaded for large media" as a final reminder that your files are at risk if you allow programs to access the large partition. It does not, however, prevent you from proceeding, assuming that if you continue, you know the risks—or know for a fact that none of your programs employ FCBs.

One way to tell whether SHARE has been loaded is to type MEM /PROGRAM at the command line and then look for it in the ensuing memory map. Programs may use the code snippet shown in Figure 1 to determine whether or not SHARE is installed. If, on return from interrupt, AL is set to FFh, then SHARE is installed; otherwise, it's not.

Here's an interesting aside: If you use SHARE in *Microsoft Windows* 3.0's 386 enhanced mode, it must be loaded before *Windows* is started. If you attempt to load it in a DOS prompt window, *Windows* will reply "SHARE already installed," even if SHARE wasn't previously installed. According to Microsoft, this is by design, not by accident. Allowing SHARE to be loaded on a virtual machine risks crashing the system (in *Windows* 386 enhanced mode, DOS applications are run in the Virtual 86 mode of the 386).

FORMATTING YOUR DISK

What's the difference between low-level formatting a hard disk and high-level formatting it?

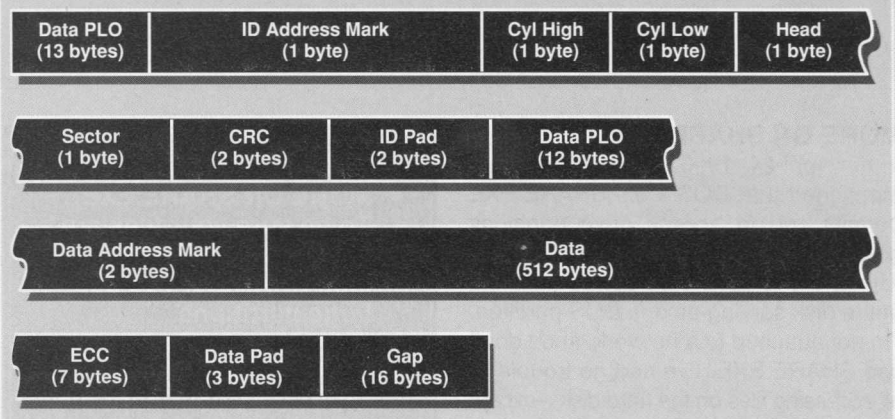
Tim Merrill
Gainesville, Florida

The basic difference between the two types of formatting is that a low-level format works at the controller level to mark off the sectors on a hard disk, while a high-level format works at the operating system level to prepare a disk for use with DOS.

Before a hard disk can be used for the first time, it must be low-level formatted. The low-level formatting process involves having the controller write information to the cylinders of the hard disk defining where the sectors are located and how they're numbered. The exact format of this information varies from controller to controller, but it typically includes a series of synchronization bytes marking the beginning of each sector, ID headers containing head, sector, and cylinder numbers, and CRC (Cyclic Redundancy Check) bytes, used to detect errors in the ID headers.

Figure 2 shows one sector format used by the popular Western Digital WD1003V-MM1 controller. Most marks, with the exception of those in the Data and ECC fields, are written to disk during the low-level formatting process.

LOW-LEVEL SECTOR FORMAT



Field	Bytes	Description
Data PLO	13	Series of sync bytes (value = 00h) marking the beginning of the sector (and of the ID field)
ID Address Mark	1	A byte (value = A1h) marking the beginning of the ID field
Cyl High	1	Indicates what range of cylinders Cyl Low corresponds to FEh: 0–255 FFh: 256–511 FCh: 512–767 FDh: 768–1023 F6h: 1024–1279 F7h: 1280–1535 F4h: 1536–1791 F5h: 1792–2047
Cyl Low	1	Low byte of cylinder number
Head	1	Head number
Sector	1	Sector number
CRC	2	Cyclical Redundancy Check bytes for detecting errors in ID field
ID Pad	2	Pad (value = AAh) for the ID field
Data PLO	12	Series of bytes (value = 00h) marking the beginning of the data field for the phase-lock oscillator
Data Address Mark	2	Bytes (values = A1h and F8h) marking the beginning of the data field
Data	512	Region where sector data is stored
ECC	7	ECC (error correction code) for fixing up bad data on reads
Data Pad	3	Pad (value = AAh) for the data field
Gap	16	Series of bytes (value = 4Eh) padding out the sector to the proper length

Figure 2: The soft-sector format used by Western Digital's WD1003V-MM1 hard disk controller. Most marks, with the exception of those in the Data and ECC fields, are written to the disk during low-level formatting.

The ID Address Mark, Cyl High, Cyl Low, Head, Sector, and CRC fields collectively form an entity known as the *sector ID*. ID Address Mark marks the beginning of the sector ID. Cyl High and Cyl Low combine to form a cylinder number ranging from 0 to 2,047, but not in the traditional least-significant byte/most-significant byte (LSB/MSB) style. Cyl Low holds the least significant 8 bits of the cylinder number; Cyl High identifies the range of cylinder numbers according to the values shown in Figure 2 (Cyl High and Cyl Low values of FDh and 10h, for example, identify cylinder number 768 plus 16, or 784). The Head and Sector fields identify the head and sector numbers, respectively. And the CRC field holds a 2-byte CRC value used to detect errors in the sector ID.

File data (the data DOS or the BIOS writes to a disk) is stored in the 512-byte Data field. The ECC field that follows holds an error-correction code that permits read errors up to a few bits in length be corrected by the controller. If not for the ECC, read errors would be far more numerous than we're accustomed to. Data fixes occur in the background, without our knowing about it. The only time DOS notifies the user about a read error is when the error is too large to be corrected by the ECC.

The ID and Data regions of the sector are preceded by Sync fields, which warn the drive's data separator that an address mark is about to pass underneath the drive head. In effect these fields allow the drive electronics to synchronize themselves with the spinning disk. At the controller's discretion, pads and gaps are added to pad out sectors (or fields within sectors) to convenient lengths.

There's more to low-level formatting than meets the eye. Even something as seemingly innocuous as sector numbering is made complex by the need to match the performance of the controller and other components of the system (the bus, DMA controllers, and so forth) to that of the hard disk itself. Sector numbering takes into account factors such as interleaving, head skew, and cylinder skew. These are all designed to keep the controller from missing a sector and having to wait for it to come back around after it moves the drive heads from one cylinder to the next, switches between heads, or finishes processing the last batch of data.

Sector interleave is the separation of consecutively numbered sectors to match the speed of the controller to the hard disk's rate of rotation (on a disk with 3:1 sector interleave, each logical sector is three physical sectors apart).

To illustrate the difference between physical and logical sectors, imagine a row of houses on a street. The first three houses might be numbered 1801, 1805, and 1807. Those are their *logical* numbers. Physically, however, they're the first, second, and third houses on the street. These are their *physical* numbers. On a disk, logical sectors are ordered in the manner that allows the controller to deliver the best performance possible, which might mean that sector 2 doesn't appear after sector 1; there may be other sectors in between.

Head skew is the number of sectors by which sectors on consecutive platters are offset to compensate for the time it takes the controller to switch from one head to the next. *Cylinder skew* is the offset between sectors on adjacent cylinders on the hard disk, included to compensate for the time it takes for a head to seek from one cylinder to the next.

The low-level format also marks bad sectors on the disk so that they won't be used. Some hard disks and controllers require defect maps shipped with the drive to be supplied (read: typed in by you) to the low-level formatting software. (A defect map is simply a list of sectors that the drive manufacturer found to be bad. It's usually just a piece of paper taped to the drive housing.)

That's okay unless the defect list happens to be really long or you lose it and later need to reformat the drive. ESDI drives store defect information right on the disk, so low-level formatting utilities for ESDI drives are able to mark the bad sectors off for you.

That's fine, too, unless additional defects develop over time as a result of normal wear and tear or head crashes. This is why disk-maintenance utilities such as *SpinRite II*, which perform low-level formats nondestructively (without destroying the files already on the disk), also do a thorough surface analysis as they format and mark off any bad sectors they find. *SpinRite II* marks the sector bad, moves the data to a good sector, and, if required, performs any necessary fix-ups to the disk's file-allocation table.

Once the low-level format is complete, all the heads, sectors, and cylinders are clearly defined, but the disk lacks the data structures DOS needs to manage files on the disk. High-level formatting, accomplished with DOS's FORMAT command or a third-party formatting utility such as the *Norton Utilities*' Safe Format or *PC Tools Deluxe*'s PC Format (both discussed in the June 26, 1990, Tutor column), sets up these data structures. They include the following:

- the *boot sector*, which contains a table of information defining the characteristics of the formatted disk (number of bytes per sector, sectors per cluster, sectors per cylinder, and so on) and the bootstrap code that reads DOS into memory on system-formatted disks and diskettes;
- the *file allocation table* (two copies), where DOS keeps track of the physical location of the files on the disk;
- the *root directory*, where filenames, directory names, and related information are stored; and
- the *data area*, where files are stored.

If you need them to, most high-level formatting utilities will also copy the DOS system files IBMBIO.COM, IBMDOS.COM, and COMMAND.COM (the first two are named IO.SYS and MSDOS.SYS in some versions of MS-DOS) to the disk and add a volume label for the purpose of identification.

Keep in mind that disk partitioning, which was discussed in the September 11, 1990, Tutor column, takes place *after* low-level formatting but *before* high-level formatting.

Partitioning divides the disk into several smaller logical disks. High-level formatting works within partition boundaries. As a result, different partitions on the same hard disk can be high-level formatted for use with different operating systems. That's why Microsoft introduced the MS-DOS partitioning scheme in the first place.

ASK THE TUTOR

The Tutor solves practical problems and explains techniques for using your hardware and software more productively. General questions about DOS and operating systems are answered here. To have your question answered, write to Tutor, *PC Magazine*, One Park Avenue, New York, NY 10016, or upload it to PC MagNet (see page 8 for access instructions). We're sorry, but we're unable to answer questions individually. ■